BÁO CÁO ĐỒ ÁN CÁ NHÂN 1

HỌ TÊN: TRẦN QUANG MINH
MSSV: 1612374

Contents

1. Linux kernel module

- Modules là những đoạn code có thể được tải vào hoặc gỡ ra từ nhân hệ điều hành dựa trên những câu lệnh. Các module này giúp mở rộng thêm các tính năng khác cho hệ điều hành mà không cần phải reboot lại hệ điều hành. Ví dụ, một loại module như device driver, nó cho phép nhân hệ điều hành có thể truy cập vào các thiết bị phần cứng và kết nối chúng với hệ thống. Nếu không xây dựng nhân hệ điều hành theo kiểu chia thành từng module, ta phải xây dựng một nhân hệ điều hành với kích thước lớn với đầy đủ các chức năng, và khi cần đến một chức năng mới (một loại device mới cần quản lý), nhân hệ điều hành sẽ phải được rebuild và reboot lại từ đầu rất khó khăn.
- Trong linux, ta có vài thao tác và lệnh cơ bản để lập trình linux kernel module như:
    - lsmod: lấy thông tin của tất cả các module đã được load vào trong kernel (các module này được lưu ở đường dẫn /proc/modules.
    - Cách module tìm đường vào trong kernel: có ai cách nhận diện một module khi module được đưa vào, đó là tên module hoặc generic indentifier (được lưu trong /etc/modprobe.config).
    - Dùng lệnh insmod <tên module> để load một module vào kernel, đôi khi load một module thành cồng thì phải load module tiên quyết trước đó.

2. Quản lý device trong linux

- Trong tất cả các loại module, device driver là một loại mà cung cấp chức năng cho hệ điều hành với các loại phần cứng thiết bị như TV card, serial port,… Trên thực tế, mỗi device đều được đại diện bởi một file trong /dev. File này cung cấp môi trường giao tiếp giữa device với người dung.

a. Major and minor number

- Tất cả các device file có cùng major number thì đều được điều khiển bởi cùng một device.
- Các device file có cùng major number thì sẽ có minor number khác nhau để phân biệt.

b. Character device and block device

- Device được chia làm 2 loại là: character device và block device.
- Sự khác biệt nằm ở việc block device chỉ có thể nhận và trả input ở dạng block (kích thước thay đổi phụ thuộc vào device), trong khi character device có thể dùng bao nhiêu byte tùy thích. Thế nên đa phần các device đều là character device.
- Khi dùng lệnh ls -l /dev/<tên device>, chữ cái đầu tiên sẽ là c nếu là character device và sẽ là b nếu là block device.
- Tạo device trong linux bằng câu lệnh: mknod /dev/<tên device> <loại dev c or b> <major> <minor>, có thể không cần <minor> vì thường thì nó sẽ được tạo tự động.

3. Giao tiếp giữa tiến trình ở user space và code kernel space.

- CPU có thể chạy ở nhiều mode khác nhau, ta gọi chúng là các ring.
- Unix dùng 2 ring, ring 0 (supervisor mode) và ring 1 (user mode).
- Thường thì chúng ta sử dụng các thư viện hàm ở user mode, những library function này sẽ gọi một hoặc nhiều system call (được thực hiện ở supervisor mode). Sau khi thực thi xong, nó sẽ trả về trở lại user space.

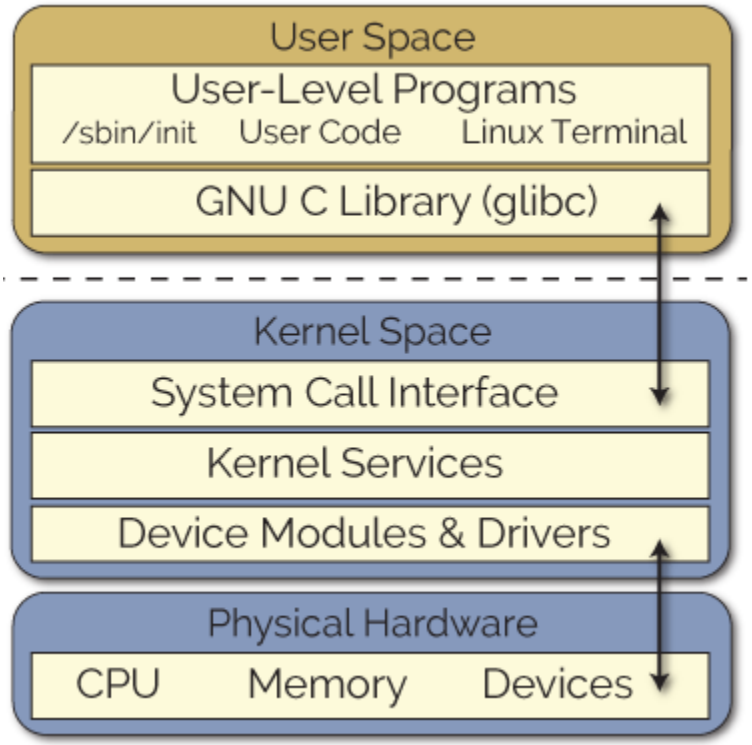

Figure 1 Linux kernel space and user space

4. Module tạo số ngẫu nhiên

- Tạo thư mục chứa project.
- Tạo file randmodule.c trong thư mục chứa project.
- Trong file randmodule.c ta có các đoạn code như sau:

- Khai báo thư viện và các thông tin liên quan:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/device.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/random.h>
```

```
#include <linux/libc-compat.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("TRAN QUANG MINH");
MODULE_DESCRIPTION("A simple Linux character device for open and read a random number");
MODULE_VERSION("0.1");
```

- Khai báo struct các operation:

```
static struct file_operations fops ={
        .open = dev_open,
        .read = dev_read,
        .release = dev_release,
};
```

- Viết các hàm open, read, release:
  - **Ở hàm open:** đếm số lần open device, kiểm tra lỗi.
  - **Ở hàm release**: thông báo kết thúc
  - **Ở hàm read**: Đây là hàm để tạo số ngẫu nhiên, gọi hàm get_random_bytes(rand, sizeof(rand)) với rand là char rand[8]. Sử dụng hàm copy_to_user(buffer, rand, sizeof(rand)) để gửi 8 bytes được random lên user space. Ở user space, ta gọi hàm read(fd, <địa chỉ số cần random>, số byte gửi qua) để lấy số random.

```c
static int dev_open(struct inode *inodep, struct file *filep)
{
    number_Opens++;
    printk(KERN_INFO "RANDMODULE: Device has been opened %d time(s)\n", number_Opens);
    return 0;
}

static ssize_t dev_read(struct file *filep, char *buffer, size_t len, loff_t *offset)
{
    char rand[8];
    get_random_bytes(rand, sizeof(rand));


    int error_count = 0;

    error_count = copy_to_user(buffer, rand, sizeof(rand));

    if (error_count == 0)
    {
        printk(KERN_INFO "MODULERAND: Sent %d characters to the user\n", sizeof(rand));
        return 1;
    }
    else
    {
        printk(KERN_INFO "MODULERAND: Failed to send %d characters to the user\n", error_count);
        return -EFAULT;
    }
}

static int dev_release(struct inode *inodep, struct file *filep)
{
    printk(KERN_INFO "RANDMODULE: Device successfully closed\n");
    return 0;
}
```

Figure 2 Dev_operation

- Gọi module_init(), module_exit():
  - Ở module_init(), ta lần lượt get major number, tạo device file, tạo device class.

```
static int __init randmodule_init(void){
    major = register_chrdev(0,DEVICE_NAME,&fops);
    if (major < 0){
        printk(KERN_ALERT"RANDMODULE: failed to register major number");
        return major;
    }

    printk(KERN_INFO"RANDMODULE: registered succesfully, major = %d", major);

    randmoduleClass = class_create(THIS_MODULE, "randmoduleClass");
    if (IS_ERR(randmoduleClass)){
        unregister_chrdev(major, DEVICE_NAME);
        printk(KERN_ALERT "Failed to register device class\n");
        return PTR_ERR(randmoduleClass);
    }

    printk(KERN_INFO "RANDMODULE: device class registered correctly\n");

    randmoduleDev = device_create(randmoduleClass, NULL, MKDEV(major, 0), NULL, DEVICE_NAME);
    if (IS_ERR(randmoduleDev))
    {
        class_destroy(randmoduleClass);
        unregister_chrdev(major, DEVICE_NAME);
        printk(KERN_ALERT "Failed to create the device\n");
        return PTR_ERR(randmoduleDev);
    }
    printk(KERN_INFO "RANDMODULE: device class created correctly\n");
    return 0;
}
```

Figure 3 randmodule_init()

  - Ở module_exit(), các công việc được thực hiện ngược lại bằng cách hủy tất cả các thao tác được tạo ra ở trên.

```
static void __exit randmodule_exit(void)
{
    device_destroy(randmoduleClass, MKDEV(major, 0));
    class_unregister(randmoduleClass);
    class_destroy(randmoduleClass);
    unregister_chrdev(major, DEVICE_NAME);

    printk(KERN_INFO "randmodule: Goodbye World");
}
```

Figure 4 randmodule_exit()